MIT 16.07
Real Time Operating Systems Lectures

- Monday's Lecture (RTOS - 16.070 Lecture 27)
  - What is an operating system?
  - Basic operating system design concepts
  - What is a Real Time Operating System (RTOS)?
  - Realtime Kernel Design Strategies (Part One)
- Wednesday's Lecture (RTOS - 16.070 Lecture 28)
  - Realtime Kernel Design Strategies (Part Two)
- Friday's Lecture (RTOS - 16.070 Lecture 29)
  - Intertask Communication

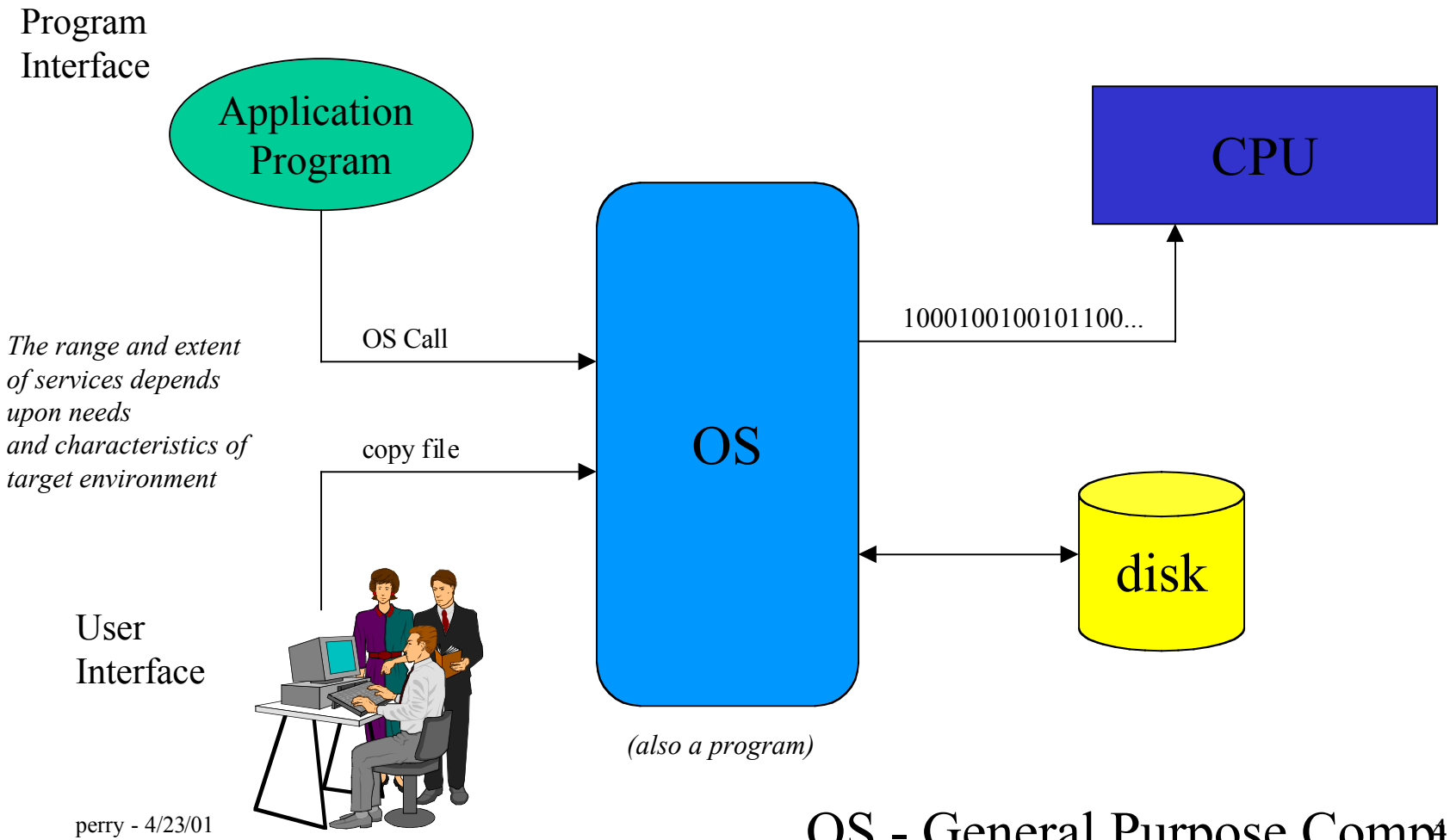
MIT 16.07 Lecture 27

Real Time Operating Systems
Part I

Operating Systems

- What is an operating system?
  - An organized collection of software extensions of hardware that serve as...
    - control routines for operating a computer (for example, to gain access to computer resources (like file I/O))
    - an environment for execution of programs

Operating System Services



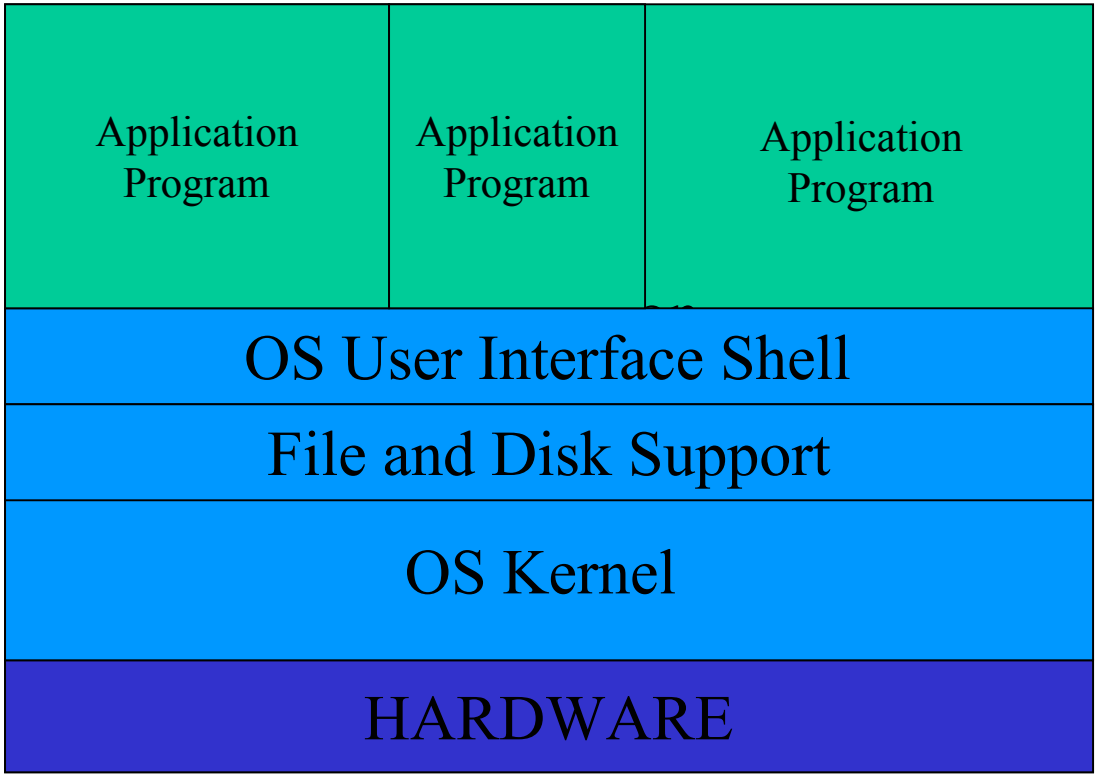
Operating Systems

- What does an operating system really do?
  - Manages computer system resources (processor, memory, I/O, etc.)
    - keeps track of status and "owner" of each resource
    - decides who gets resource
    - decides how long the resource can be in use
  - In systems that support concurrent execution of programs, it
    - resolves conflicts for resources
    - optimizes performance given multiple users
  - Think of it as the keeper of a single copy of a book that everyone in this course needs to read
    - What are some of the issues that arise?

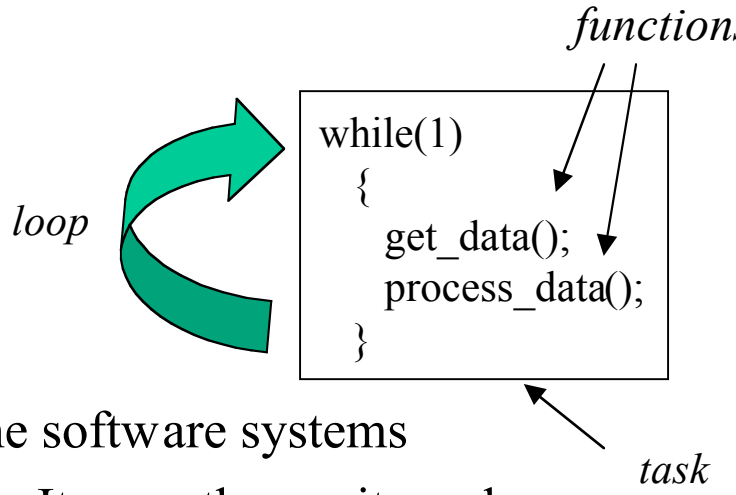
Operating Systems

- Types of Operating Systems
  - Simplest = small kernel on embedded processor
  - Most Complex = Full featured commercial OS
    - Multi-user security
    - Graphics support
    - Networking support
    - Peripherals communication
    - Concurrent execution of programs

Operating System Hierarchy



Tasks & Functions



- A task is a process that repeats itself
  - Loop forever
  - Essential building block of real time software systems
- A function is a procedure that is called. It runs, then exits and may return a value. For example,
  - process_data();
  - int add_two_numbers(int x, int y);

Operating Systems

- In most cases, RTOS = OS Kernel
  - An embedded system is designed for a single purpose so the user shell and file/disk access features are unnecessary
  - RTOS gives you control over your resources
    - No background processes that “just happen”
    - Bounded number of tasks
  - RTOS gives you control over timing by allowing…
    - Manipulation of task priorities
    - Choice of scheduling options

Operating Systems

- OS Kernel - 3 functions
  - Task Scheduler : To determine which task will run next in a multitasking system
  - Task Dispatcher: To perform necessary bookkeeping to start a task
  - Intertask Communication: To support communication between one process (i.e. task) and another

The OS Kernel

- Going back to the book analogy
  - Task Scheduler: Who gets the book and when?
  - Task Dispatcher: Handling the logistics of getting the book from one person and giving it to another
  - Intertask Communication: What if one student wanted to talk with another? Only one student can have the book at one time.

Types of RTOS

Realtime Kernel Design Strategies

- Polled Loop Systems *(today's lecture)*
- Interrupt Driven Systems *(today's lecture)*
- Multi-tasking *(4/25 lecture)*
- Foreground / Background Systems *(4/25 lecture)*
- Full Featured RTOS *(4/25 lecture)*

Types of RTOS

NERF BALL DEMO

A look at some real time operating system issues
illustrated through a simple tossing of nerf balls

Polled Loops

- Simplest RT kernel
- A single and repetitive instruction tests a flag that indicates whether or not an event has occurred.
- No intertask communication or scheduling needed. Only a single tasks exists.
- Excellent for handling high-speed data channels, especially when
    - events occur at widely spaced intervals and
    - processor is dedicated to handling the data channel

Polled Loop Example

Identification Friend or Foe (IFF) system software communicates with a central alarm processor in an aircraft cockpit. If a contact is made, a flag called "IFF_data_here" is set by the network and the alarm software processes the data.

```
while (TRUE)                          /* infinite loop, do forever */
    {
    if IFF_data_here==TRUE then       /* check for IFF data */
          {
          process_data();             /* call process_data() function*/
          IFF_data_here=FALSE;        /* reset flag */
          }
    }
```

An Extension of the Polled Loop Example

```
int contact;
int get_IFF_data();
void sound_alarm();
void log_contact();
int red = 1;
int blue = 2;
while (1)
        {
        contact = get_IFF_data();                       /* get data from IFF software*/
        switch (contact)
                {
                case red:       {
                                sound_alarm();          /* sound alarm in cockpit */
                                break;
                                }
                case blue:      {
                                log_contact();          /* put contact data in memory */
                                break;
                                }
                }   /* end switch */
        }  /* end infinite loop */
```

Polled Loops

- Pros:
  - Simple to write and debug
  - Response time easy to determine
- Cons:
  - Can fail due to burst of events
  - Generally not sufficient to handle complex systems
  - Waste of CPU time, especially when event being polled occurs infrequently

Using Polled Loops

- Often used inside other real time schemes to, for example,
  - poll a suite of sensors for data
  - check for user inputs (keyboard or keypad data)
- Opposite of interrupt driven systems

MIT 16.07 - RTOS

Realtime Kernel Design Strategies

- Polled Loop Systems
- **Interrupt Driven Systems**
- Multi-tasking
- Foreground / Background Systems
- Full Featured RTOS

What is an Interrupt?

- A hardware signal that initiates an event
- Upon receipt of an interrupt, the processor:
  - completes the instruction being executed
  - save the program counter (so as to return to the same execution point)
  - loads the program counter with the location of the interrupt handler code
  - executes the interrupt handler
- In practice, real time systems can handle several interrupts in priority fashion
  - Interrupts can be enabled/disabled
  - Highest priority interrupts serviced first

INTERRUPT-DRIVEN SYSTEMS

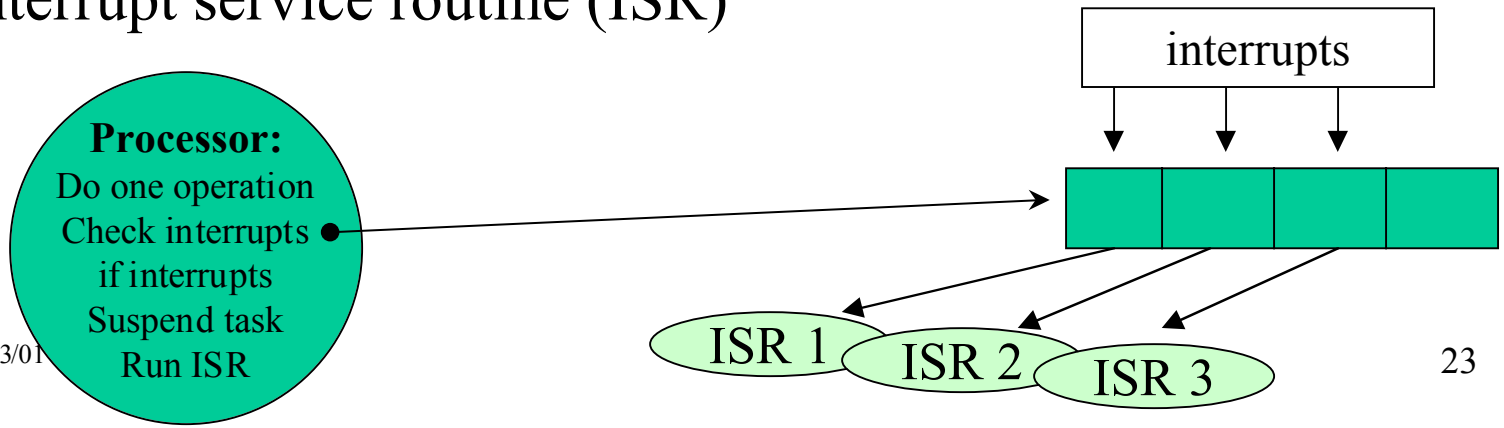
- Interrupt Driven Software Examples
  - IFF receiver sees a threat and interrupts an aircraft mission computer to sound a cockpit alarm
  - Inertial Navigation Unit data (Δ velocities in north/east/up coordinates) is available at 32 Hz and interrupts the navigation software with new data when it is ready
  - Sonar contact data interrupts signal processing software when new data is available
  - Low altitude indicator triggers a fly-up command for a pilot

Additions to Lecture 27

- The following two slides contain additional information about interrupts and interrupt handling as discussed in lecture #27

Interrupt Handling

- An interrupt is a software or hardware signal to the processor
  - Indicates something urgent is happening
    - Current task wants to sleep or get I/O
    - Scheduler wants to run a different task now
    - Mouse just moved or keyboard key was struck
    - Sensors detect inbound hostile weapons
- Processor must check for interrupts very frequently. If any have arrived, it stops immediately and runs the associated interrupt service routine (ISR)



Interrupt Service Routine

- A program run in response to an interrupt
  - Disables all interrupts
  - Runs code to service the event
  - Clears the interrupt flag that got it called
  - Re-enables interrupts
  - Exits so the processor can go back to its running task
- Should be as fast as possible, because nothing else can happen when an interrupt is being serviced.
- Interrupts can be:
  - Prioritized (service some interrupts before others)
  - Disabled (processor doesn't check or ignores all of them)
  - Masked (processor only sees some interrupts

MIT 16.07 - RTOS Lecture 27 Summary

- An operating system is a software extension of the hardware in a computer
  - program interface
  - user interface
- An operating system manages computer system resources
- A real time operating system is often just the OS kernel (i.e. no fancy features, no user interface). Just...
  - task scheduler
  - task dispatcher
  - intertask communcation
- A task is an infinite loop with a real time purpose

MIT 16.07 - RTOS Lecture 27 Summary

- There are several Realtime Kernel Design Strategies. These include...
  - Polled loop systems (section 6.1 in your Real Time text)
    - A single and repetitive instruction tests a flag that indicates whether or not an event has occurred.
    - Simplest option
  - Interrupt Driven Systems (Section 6.4)
    - Processing continues until interrupted by external events
    - After interrupt has been serviced, processing resumes where it left off

- Next lecture we will discuss multitasking and work our way on up to a full featured RTOS (Sections 6.5, 6.6)
- Friday we will talk about communication among tasks (Sections 7.1-7.6)